UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|---|---|---|---|---|
| 10/567,948 | 02/10/2006 | Alan Walter Stiemens | 0231 | 4652 |

31665 7590 01/04/2010

PATENT DEPARTMENT
MACROVISION CORPORATION
2830 DE LA CRUZ BLVD.
SANTA CLARA, CA 95050

| EXAMINER |
|---|
| CHOWDHURY, ZIAUL A. |

| ART UNIT | PAPER NUMBER |
|---|---|
| 2192 | |

| MAIL DATE | DELIVERY MODE |
|---|---|
| 01/04/2010 | PAPER |

**Please find below and/or attached an Office communication concerning this application or proceeding.**

The time period for reply, if any, is set in the attached communication.

| Office Action Summary | Application No. | Applicant(s) | |
|---|---|---|---|
| | 10/567,948 | STIEMENS ET AL. | |
| | Examiner | Art Unit | |
| | ZIAUL CHOWDHURY | 2192 | |

*-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --*

**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

1)☒ Responsive to communication(s) filed on *10 February 2006*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

4)☒ Claim(s) *67-105* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *67-105* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

**Application Papers**

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. § 119**

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *02/10/2006*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

Art Unit: 2192

## Detailed Action

1. This is the initial office action bases on the preliminary amendment filed on February 10th, 2006, wherein claims 1-66 has been cancelled, and claims 67 to 105 are presented for examination.

## Status of Claims

2. Claims 67-105 are pending, of which claims 67, 73, 78, 87, 97, and 101 are in independent form.

## Oath/Declaration

3. The office acknowledges receipt of a properly signed oath/declaration filed on February 10th, 2006.

## Priority

4. The priority date that has been considered for this application is August 19th, 2004.

## Information Disclosure Statement

5. The information disclosure statements filed on February 10th, 2006 comply with the provisions of 37 CFR 1.97, 1.98. They have been placed in the application file and the information referred to therein has been considered as to the merits.

## Claim Rejections - 35 USC § 101

6. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

7. Claims 73-77 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

Claim 73 involving software program encoded with functional descriptive material do not fall within any of the categories of patentable subject matter set forth in 35 U.S.C. § 101, and such claim is therefore ineligible for patent protection. Note that the phrase "An executable program in object code" in the claim which is directed to a program itself. A program can be computer software alone which means it is directed to a process, not directed to machine, manufacture or produces a physical transformation moreover, the claim itself does not disclose any hardware component to realize any of the underlying functionality of the intended process. Because they lack physical structure, program it self *per se* does not fall within one of the four statutory categories of invention (See Lowry, 32 F.3d at 1583-84, 32 USPQ2d at 1035). Therefore, it is clear that claim 73 does not fall within any of the categories of patentable subject matter set forth in 35 U.S.C. § 101, and such claim is therefore ineligible for patent protection.

Claims 74-77 do not overcome the deficiencies as noted above for claim 73; therefore claims 74-77 are also rejected as non-statutory subject matter.

## Claim Rejections - 35 USC § 102

> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another

> filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

8. Claims 67-96 are rejected under 35 U.S.C. 102(e) as being anticipated by Chen et al. (US Patent Publication No 2004/0003278 –art being made of record)

In reference to claim 67:

Chen discloses :

A method of producing obfuscated object code, the method comprising:

*substituting a variable in source code with a selected function of the variable,*

(Paragraph 0016; The OTL selection module randomly selects or generates one of the possible variable obfuscation functions for each declared secure variable. The OTL substitution module substitutes the separate instance of the selected variable obfuscation function for every reference to the declared secure variable.)

*and compiling the source code to produce object code,*

(Paragraph 0057; The compiler module 712 processes the source code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module.)

*wherein the selected function is arranged to cause the variable to be presented in the compiled object code as a series of operations.*

(Paragraph 0016; The system includes an OTL selection module, an OTL substitution module, an OTL type library database, a compiler module; and a linker module to create an executable processing module. The OTL selection module randomly selects or generates one of the possible variable obfuscation functions for each declared secure variable.)

Art Unit: 2192

In reference to claim 68:

Chen discloses :

*wherein the series of operations by which the variable is presented is made up of arithmetic and/or logical operations,*

(Paragraph 40; It is known how (by carefully composing elementary arithmetic operations with random constants) to generate authentication tag generation mechanisms (reversible and non reversible) randomly and they may be used here to automatically choose encryption and authentication mechanisms. One of our ideas is to transform an OTL Boolean object that serves as a branch condition into a number of objects (logical), which are further checked by code in both targets of the branch. There are exponential number of ways to obfuscate a Boolean variable and branch conditions.)

*and wherein the series of operations is arranged, upon running of the object code, to provide the variable.*

(Paragraph 48; The OTL processing of this program module 300 will create a separate instance of a function that will obfuscate the contents of these secure variables. For any given type of variable, i.e., integer, float, character, Boolean, etc., a set of possible functions may be included within a library for use at compile time. Alternately, these functions may be generated on-the-fly during compile time.)

In reference to claim 69:

Chen discloses :

*wherein the series of operations by which the variable is presented comprises complementary operations arranged, upon running of the object code, to provide the variable.*

(Paragraph 57; The compiler module 712 processes the source code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module

713 to generate an executable processing module.)

In reference to claim 70:

Chen discloses :

*wherein the source code involves stored templates and the selected function is defined in a template of the source code,*

(Refer to FIG 3, and associated text)

*the template of the source code defining a plurality of functions which are each arranged to compile to present the variable as a series of operations,*

(Refer to FIG4, section 302, section 303, and section 304; and associated text)

*the method further comprising selecting one of the functions to substitute for the variable in the source code.*

(Paragraph 52; another embodiment of the present invention will substitute the code for the secure function directly into the in-line code of the main procedure section 303.)

In reference to claim 71:

Chen discloses :

*wherein a different key is associated with each one of the functions in the template,*

(Paragraph 36; the OTL compiler component is free to choose (using a random key) whichever meta-implementation is available for a given OTL type and individualize the type instantiation on a per-object basis.)

*and the method further comprises substituting the variable in source code with the template and selecting one of the functions in the template by selecting the key which is associated with said one function.*

(Paragraph 36; the OTL compiler component is free to choose (using a random key) whichever meta-implementation is available for a given OTL type and individualize the type instantiation on a per-object basis.)

In reference to claim 72:

Chen discloses :

*wherein the source code is C++.*

(Paragraph 38; OTL is tightly integrated into the programming environment either through a rich class hierarchy in the model of MFC and ATL, or by augmented C/C++/C# languages and compilers.)

In reference to claim 73:

Chen discloses :

*An executable program in object code, the program having been compiled from source code including variables, wherein a variable in the source code has been compiled to be presented as a series of operations in object code whereby the object code is obfuscated.*

(Refer to Fig 9, and Fig 10,

Paragraph 0027 FIG. 9 illustrates a processing system for generating executable program modules using OTL modules according to another possible embodiment of the present invention.

Paragraph 0067; When module 1021 finds a reference, module 1022 substitutes the previously created instance of the obfuscation function for the referenced secure variable into the source code. Test module 1023 determines if any additional secure variables references exist within the source code. If additional secure variable references are found the processing returns to module 1021 to process the next secure variable reference. If test module 10223 determines no additional secure variables references are to be processed, the processing continues with module 1024 to optimize, compile, link and further obfuscate the variables as discussed above. Once the executable processing module is created, the processing ends 1002.)

In reference to claim 74:

Chen discloses :

*wherein the series of operations by which the variable is presented is made up of arithmetic and/or logical operations,*

(Paragraph 40; It is known how (by carefully composing elementary arithmetic operations with random constants) to generate authentication tag generation mechanisms (reversible and non reversible) randomly and they may be used here to automatically choose encryption and authentication mechanisms. One of our ideas is to transform an OTL Boolean object that serves as a branch condition into a number of objects (logical), which are further checked by code in both targets of the branch. There are exponential number of ways to obfuscate a Boolean variable and branch conditions.)

*and wherein the series of operations is arranged, upon running of the object code, to provide the variable.*

(Paragraph 48; The OTL processing of this program module 300 will create a separate instance of a function that will obfuscate the contents of these secure variables. For any given type of variable, i.e., integer, float, character, Boolean, etc., a set of possible functions may be included within a library for use at compile time. Alternately, these functions may be generated on-the-fly during compile time.)

In reference to claim 75:

Chen discloses :

*wherein the series of operations by which the variable is presented comprises complementary operations arranged, upon running of the object code, to provide the variable.*

(Paragraph 57; The compiler module 712 processes the source code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module.)

In reference to claim 76:

Chen discloses :

*wherein the series of operations has been produced by substituting the variable in the source code with a selected function arranged to cause the variable to be presented in the compiled object code as a series of operations.*

(Refer to FIG4, section 302, section 303, and section 304; and associated text)

In reference to claim 77:

Chen discloses :

*wherein the source code involved stored templates and the selected function was defined in a template of the source code,*

(Refer to FIG 3, and associated text)

*and the template of the source code had defined a plurality of functions which were each arranged to compile to present the variable as a series of operations,*

(Refer to FIG4, section 302, section 303, and section 304; and associated text)

*and wherein one of the functions had been selected to substitute for the variable in the source code.*

(Paragraph 52; another embodiment of the present invention will substitute the code for the secure function directly into the in-line code of the main procedure section 303.)

In reference to claim 78:

Chen discloses :

*A method of producing storage media having a secured executable program thereon, the method comprising the steps of securing an*

*executable program by associating the executable program with a security program which is arranged to control access to the executable program, and applying the secured executable program to a storage media,*

(Page 8, Col2:38-42; A computer program data product readable by a computing system and encoding instructions implementing a method for providing secure and opaque type libraries to automatically provide secure variables within a programming module,)

*and the method further comprising obfuscating the object code of the security program,*

(Paragraph 17; The method identifies secure variable declaration statements within a source code module, queries a database for identification of possible variable obfuscation functions applicable for the variable type corresponding to the secure variable declaration statement, randomly selects or generates one of the possible variable obfuscation functions returned in response to the database query, creates a separate instance of the selected variable obfuscation function corresponding to the secure variable declaration, and substitutes the separate instance of the selected variable obfuscation function for every reference to the declared secure variable.)

*wherein the object code of the security program has been obfuscated by substituting a variable in source code with a selected function of the variable,*

(Paragraph 17; The method identifies secure variable declaration statements within a source code module, queries a database for identification of possible variable obfuscation functions applicable for the variable type corresponding to the secure variable declaration statement, randomly selects or generates one of the possible variable obfuscation functions returned in response to the database query, creates a separate instance of the selected variable obfuscation function corresponding to

the secure variable declaration, and substitutes the separate instance of the selected variable obfuscation function for every reference to the declared secure variable.)

*and compiling the source code to produce object code, the selected function causing the variable to be presented in the compiled object code as a series of operations.*

(Paragraph 0057; The compiler module 712 processes the source code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module.)

In reference to claim 79:

Chen discloses :

*wherein the executable program and the security program are associated at object code level,*

(Paragraph 57; The compiler module 712 processes the source code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module.)

*the security program being arranged to encrypt the executable program.*

(Paragraph 16; One aspect of the present invention is a system for providing secure and opaque type libraries to automatically provide secure variables within a programming module (encrypt). The system includes an OTL selection module, an OTL substitution module, an OTL type library database, a compiler module; and a linker module to create an executable processing module.)

In reference to claim 80:

Chen discloses :

*further comprising moving blocks of the executable program out of the executable program and relocating the blocks in the security program.* (Paragraph 57; The compiler module 712 processes the source code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module.)

In reference to claim 81:

Chen discloses :

*wherein the security program is arranged to require the running of an authentication program.*

(Paragraph 40; It is known how (by carefully composing elementary arithmetic operations with random constants) to generate authentication tag generation mechanisms (reversible and non reversible) randomly and they may be used here to automatically choose encryption and authentication mechanisms.)

In reference to claim 82:

Chen discloses :

*wherein the source code of the security program involves stored arrays and templates and utilises pointers to navigate the arrays and templates.* (Paragraph 47; In the example, a programming module 300 consists of a set of sections that include a header 301, a variable declaration section 302, a main procedure section 303, and a methods and functions section 304. The header section 301 may be used to provide needed declarations and metadata needed to specify the compile-time and run-time parameters of the programming module once an executable module is created from a source code file. The variable declaration section 302 allows for the declaration of any variables, especially secure variable that may utilize OTL libraries according to the present invention. The main

procedure module 303 provides the set of instructions that are performed when the programming module is run. Within the main procedure module, various instructions 335-336 may reference variables declared in the variable declaration section. Finally, the methods and functions section 304 provides any additional set of functions and methods that may be utilized by the instructions within the main procedure module 303 as defined by a programmer.)

In reference to claim 83:

Chen discloses :

*wherein the source code of the security program is C++*

(Paragraph 38; OTL is tightly integrated into the programming environment either through a rich class hierarchy in the model of MFC and ATL, or by augmented C/C++/C# languages and compilers.)

In reference to claim 84:

Chen discloses :

*wherein the storage media onto which the secured executable program is applied is an optical disc.*

(Paragraph 44; A number of program modules may be stored on the hard disk, magnetic disk 216, optical disk 219, ROM 208 or RAM 210, including an operating system 226, one or more application programs 228, other program modules 230, and program data 232.)

In reference to claim 85:

Chen discloses :

*wherein the secured executable program is applied to the optical disc by laser beam encoding.*

(Paragraph 42; The computer system 200 further includes a hard disk drive 212 for reading from and writing to a hard disk, a magnetic disk

drive 214 for reading from or writing to a removable magnetic disk 216, and an optical disk drive 218 for reading from or writing to (done by laser beam encoding) a removable optical disk 219 such as a CD ROM, DVD, or other optical media –emphasis added.)

In reference to claim 86:

Chen discloses :

*wherein the storage media onto which the secured executable program is applied is memory in, or associated with, servers, computers and/or other processing means.*

(Paragraph 41; an exemplary computing system for embodiments of the invention includes a general purpose computing device in the form of a conventional computer system 200, including a processor unit 202, a system memory 204, and a system bus 206 that couples various system components including the system memory 204 to the processor unit 200. Paragraph 45; The computer system 200 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 246. The remote computer 246 may be a computer system, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer system 200.)

In reference to claim 87:

Chen discloses :

*A storage media having a secured executable program thereon, wherein an executable program is secured by having a security program associated therewith,*

(Page 8, Col2:38-42; A computer program data product readable by a computing system and encoding instructions implementing a method for

providing secure and opaque type libraries to automatically provide secure variables within a programming module,)

*the security program being arranged to control access to the executable program,*

(Page 8, Col 2:5-9; compiling the source code; linking the source code with additional processing modules; and applying additional obfuscation processing to protect data-memory access patterns.)

*and wherein the security program is in object code which has been obfuscated,*

(Paragraph 0057; The compiler module 712 processes the source code following the OTL code (obfuscated code) substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module – emphasis added.)

*the security program having been compiled from source code including variables,*

(Paragraph 0015; The present invention relates to a method, apparatus, and article of manufacture to provide secure variables within programming modules and more particularly to a system, method and article of manufacture for providing secure and opaque type libraries to automatically provide secure variables within a programming module.)

*and a variable in the source code of the security program having been compiled to be presented in object code as a series of operations whereby the object code has been obfuscated.*

(Paragraph 0053; When the source code is processed as part of a compilation of the processing module 300, the secure variable declaration statements 522 - 524 will cause instances of the secure functions to be generated as discussed above. For each variable type, a particular instance of an obfuscation function is randomly selected or generated, as are any parameters utilized by the selected function. When

the main procedure section 303 is processed, these instances of obfuscation functions are substituted into the particular instructions that reference a particular secure variable.
Paragraph 0057; The compiler module 712 processes the source code following the OTL code (obfuscated code) substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module – emphasis added.)

In reference to claim 88:
Chen discloses :
*wherein the series-of-operations-by which the variable-is presented is made up of arithmetic and/or logical operations,*
(Paragraph 40; It is known how (by carefully composing elementary arithmetic operations with random constants) to generate authentication tag generation mechanisms (reversible and non reversible) randomly and they may be used here to automatically choose encryption and authentication mechanisms. One of our ideas is to transform an OTL Boolean object that serves as a branch condition into a number of objects (logical), which are further checked by code in both targets of the branch. There are exponential number of ways to obfuscate a Boolean variable and branch conditions.)
*and wherein the series of operations is arranged, upon running of the object code, to provide the variable.*
(Paragraph 48; The OTL processing of this program module 300 will create a separate instance of a function that will obfuscate the contents of these secure variables. For any given type of variable, i.e., integer, float, character, Boolean, etc., a set of possible functions may be included within a library for use at compile time. Alternately, these functions may be generated on-the-fly during compile time.)

In reference to claim 89:

Chen discloses :

*wherein the series of operations by which the variable is presented comprises complementary operations arranged, upon running of the object code, to provide the variable.*

(Paragraph 57; The compiler module 712 processes the source code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module.)

In reference to claim 90:

Chen discloses :

*wherein the executable program and the security program are associated at object code level,*

(Paragraph 57; The compiler module 712 processes the source code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module.)

*and wherein the executable program is encrypted on the storage media and the associated security program enables decryption of the executable program.*

(Paragraph 60; A TW application written with OTL is compiled into two components: the application binary and an OTL object decoder (OTL decoder) as shown in FIG. 8a. The OTL decoder contains necessary information for someone to de-obfuscate (decryption) the OTL objects for this application.)

In reference to claim 91:

Chen discloses :

*wherein blocks from the executable program have been relocated within the security program.*
(Paragraph 57; The compiler module 712 processes the source code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module.)

In reference to claim 92:
Chen discloses :
*wherein the security program is arranged to require the running of an authentication program.*
(Paragraph 40; It is known how (by carefully composing elementary arithmetic operations with random constants) to generate authentication tag generation mechanisms (reversible and non reversible) randomly and they may be used here to automatically choose encryption and authentication mechanisms.)

In reference to claim 93:
Chen discloses :
*wherein the storage media is an optical disc on which the executable program and the security program are encoded.*
(Paragraph 44; A number of program modules may be stored on the hard disk, magnetic disk 216, optical disk 219, ROM 208 or RAM 210, including an operating system 226, one or more application programs 228, other program modules 230, and program data 232.)

In reference to claim 94:
Chen discloses :
*wherein the optical disc is one of: a CD, a CD-ROM, and a DVD.*

(Paragraph 42; The computer system 200 further includes a hard disk drive 212 for reading from and writing to a hard disk, a magnetic disk drive 214 for reading from or writing to a removable magnetic disk 216, and an optical disk drive 218 for reading from or writing to a removable optical disk 219 such as a CD ROM, DVD, or other optical media.)

In reference to claim 95:

Chen discloses :

*wherein the storage media is memory in, or associated with, servers, computers and/or processing means and on which the executable program and the security program are stored.*

(Paragraph 41; an exemplary computing system for embodiments of the invention includes a general purpose computing device in the form of a conventional computer system 200, including a processor unit 202, a system memory 204, and a system bus 206 that couples various system components including the system memory 204 to the processor unit 200. Paragraph 45; The computer system 200 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 246. The remote computer 246 may be a computer system, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer system 200.)

In reference to claim 96:

Chen discloses :

*wherein the executable program is one or more of: a games program; a video program; an audio program; and other software.*

(Paragraph 44; Examples of other input devices may include a microphone, joystick, game pad, satellite dish, and scanner. For hand-held devices and tablet PC devices, electronic pen input devices may also

be used. These and other input devices are often connected to the processing unit 202 through a serial port interface 240 that is coupled to the system bus 206. Nevertheless, these input devices also may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB).)

## Claim Rejections - 35 USC § 102

> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

9. Claim 97 is rejected under 35 U.S.C. 102(b) as being anticipated by Sokolov et al. (US Patent Publication No 2002/0194243 –art being made of record)

In reference to claim 97:

Sokolov discloses:

A method of controlling a processor to run a program comprising:

*translating instructions from the program into a reduced instruction set format to which said processor is not responsive,*

(Paragraph 15; the operations performed by conventional instructions can be performed by relatively fewer inventive virtual machine instructions. Furthermore, the inventive virtual machine instructions can be used to perform operations that cannot readily be performed by conventional Java Bytecode instructions.

Paragraph 13; Java interpreters are needlessly complex since they need to recognize a relatively large number of Java instructions and possibly implement various mechanisms for executing many instructions. Thus,

the conventional Java Bytecode instruction set is not a very desirable solution for systems with limited resources (e.g., embedded systems) Paragraph 14; Accordingly, there is a need for alternative instructions suitable for execution in virtual machines –emphasis added.)

*causing the translated instructions to be applied to a virtual processor which is responsive to the reduced instruction set format,*

(Paragraph 17; As a set of virtual machine instructions suitable for execution in a virtual machine, one embodiment of the invention provides a set of virtual machine instructions representing a number of corresponding Java Bytecode executable instructions that are also suitable for execution in the virtual machine.)

*and causing the virtual processor to run the instructions applied thereto and to apply a series of simple instructions, to which the processor is responsive, to the processor.*

(Paragraph 19; As a Java Bytecode instruction translator, one embodiment of the invention operates to convert a set of Java Bytecode executable instructions suitable for execution on a virtual machine into a set of corresponding executable virtual machine instructions. The corresponding virtual machine instructions are also suitable for execution in the virtual machine and represent operations that can be performed by execution of a number of corresponding Java Bytecode instructions. In addition, the corresponding set of the virtual machine instructions consists of a number of virtual machine instructions that is less than the number of the corresponding Java Bytecode executable instructions (reduced instruction set) –emphasis added.)

## Claim Rejections - 35 USC § 103

10. Claims 98-100 are rejected under 35 U.S.C. 103(a) as being unpatentable over Sokolov et al. in view of Lambert Martin et al (US

Art Unit: 2192

Patent Application Publication No 2004/0039926 –art being made of record).

In reference to claim 98:

Sokolov does not disclose:

further comprising encrypting the translated instructions to be applied to the virtual processor, and enabling the virtual processor to respond to the encrypted instructions without decrypting them.

However, Lambert discloses:

*further comprising encrypting the translated instructions to be applied to the virtual processor,*

(Refer to Fig 3.

Paragraph 25; Java source code 302 is compiled in step 304 to obtain bytecode 306 in the manner previously described. Thereafter, as shown by step 308, bytecode obfuscation is performed to make the bytecode more difficult to understand by removing as much helpful information as possible from the bytecode (e.g. renaming variables to cryptic strings) and making changes to the operational bytecode to make it harder to read without changing the actual operations (e.g. unrolling loops, optimising away variables, etc.). This results in an obfuscated Java bytecode 310. When maliciously decompiled, as shown by step 312, the decompiled source code 302' is thus difficult to read, i.e. obfuscated.)

*and enabling the virtual processor to respond to the encrypted instructions without decrypting them.*

(Refer to Fig 2 and associated text.)

It would have been obvious to one ordinary skill in art at the time the invention was made to combine Lambert's teaching with Sokolov's method because that would further enhance Sokolov's bytecode processing method with temper resisting capability by using bytecode obfuscating and bytecode encrypting procedure within the java virtual

machine -see Lambert paragraph 23.

In reference to claim 99:

Sokolov discloses:

*a selected template providing a series of instructions in the reduced instruction set format for each instruction from the program,*

(Paragraph 17; As a set of virtual machine instructions suitable for execution in a virtual machine, one embodiment of the invention provides a set of virtual machine instructions representing a number of corresponding Java Bytecode executable instructions that are also suitable for execution in the virtual machine. The set of the virtual machine instructions consists of a number of virtual machine instructions which is less than the number of the corresponding Java Bytecode executable instructions(reduced instruction set format).)

*wherein the templates define a plurality of series of instructions in the reduced instruction set format for an instruction in the program,*

(Paragraph 17; every one of the corresponding Java Bytecode executable instructions can be represented by at least one of the virtual machine instructions in the virtual machine instruction set.)

*the method further comprising selecting one of said plurality of series of instructions to be the translation for said instruction.*

(Paragraph 18; receiving one or more bytes representing a Java Bytecode instruction suitable for execution in a virtual machine; selecting a corresponding virtual machine instruction.)

Sokolov does not disclose:

further comprising utilizing templates to translate and encrypt the instructions,

However, Lambert discloses:

*further comprising utilizing templates to translate and encrypt the instructions,*

(Paragraph 73; A DRM encryption utility program that understands the Java archive format 410 is given the names of the classes 414a that are to be encrypted. As shown by step 412, it parses the Java archive 410 to locate the specified classes 414a, encrypts them within the Java archive and, if necessary, updates any internal archive data structures to ensure that the archive remains valid.)

It would have been obvious to one ordinary skill in art at the time the invention was made to combine Lambert's teaching with Sokolov's method because that would provide an enhanced feature to encrypt the bytecode and process through the Java Virtual machine without any further modification of the encrypted source code to be executed –see Lambert paragraph 70.

In reference to claim 100:

Sokolov does not disclose:

wherein a different key is associated with each one of the plurality of series of instructions in the reduced instruction set format in the template, and wherein the method further comprises selecting a key which is associated with one of said plurality of series of instructions and translating the instruction in the program to the one of said plurality of series of instructions which is associated with the selected key.

However, Lambert discloses:

*wherein a different key is associated with each one of the plurality of series of instructions in the reduced instruction set format in the template*

(Paragraph 109; Digital signing involves calculating cryptographic hash values for the class files within the archive and encrypting them using a secret key.)

*and wherein the method further comprises selecting a key which is associated with one of said plurality of series of instructions and*

*translating the instruction in the program to the one of said plurality of series of instructions which is associated with the selected key.*
(Paragraph 109; Digital signing involves calculating cryptographic hash values for the class files within the archive and encrypting them using a secret key. Hash functions are mathematical algorithms used to calculate a value encoded as a small number of bytes from a buffer containing a potentially far larger number of bytes.)
It would have been obvious to one ordinary skill in art at the time the invention was made to combine Lambert's teaching with Sokolov's method which would provide Sokolov's method to encrypt the small instruction set data using secret key before the bytecode is processed by Java Virtual machine which in turn will provide the consumer to access only the selective contend which is intended to be consumer's usage parameter –see Lambert paragraph 003.

## Claim Rejections - 35 USC § 103

11. Claims 101-105 are rejected under 35 U.S.C. 103(a) as being unpatentable over Chen et al. in view of Sokolov et al.

In reference to claim 101:
Chen discloses:
*A storage media having a secured executable program thereon, wherein an executable program is secured by having a security program*
(Page 8, Col2:38-42; A computer program data product readable by a computing system and encoding instructions implementing a method for providing secure and opaque type libraries to automatically provide secure variables within a programming module,)
*the security program being arranged to control access to the executable program,*

(Page 8, Col 2:5-9; compiling the source code; linking the source code with additional processing modules; and applying additional obfuscation processing to protect data-memory access patterns.)

Chen does not disclose:

and an emulation program associated therewith, and the emulation program causing predetermined functions or routines of the executable program to be run on a virtual processor provided by said emulation program, wherein the emulation program is arranged to translate instructions from the executable program into a reduced instruction set format, to cause the translated instructions to be applied to the virtual processor, and to cause the virtual processor to run the instructions applied thereto and to output a series of simple instructions for application to a processor.

However, Sokolov discloses:

*and an emulation program associated therewith, and the emulation program causing predetermined functions or routines of the executable program to be run on a virtual processor provided by said emulation program,*

(Paragraph 15; The inventive virtual machine instructions can effectively represent the complete set of operations performed by the conventional Java Bytecode instruction set. Moreover, the operations performed by conventional instructions can be performed by relatively fewer inventive virtual machine instructions. Furthermore, the inventive virtual machine instructions can be used to perform operations that cannot readily be performed by conventional Java Bytecode instructions. Thus, a more elegant yet robust virtual machine instruction set can be implemented. This in turn allows implementation of relatively simpler interpreters as well as allowing alternative uses of the limited 256 (2.sup.8) Bytecode representation (e.g., a macro representing a set of commands). As a

result, the performance of virtual machines, especially, those operating in systems with limited resources, can be improved.)

*wherein the emulation program is arranged to translate instructions from the executable program into a reduced instruction set format, to cause the translated instructions to be applied to the virtual processor, and to cause the virtual processor to run the instructions applied thereto and to output a series of simple instructions for application to a processor.*

(Paragraph 17; As a set of virtual machine instructions suitable for execution in a virtual machine, one embodiment of the invention provides a set of virtual machine instructions representing a number of corresponding Java Bytecode executable instructions that are also suitable for execution in the virtual machine. The set of the virtual machine instructions consists of a number of virtual machine instructions which is less than the number of the corresponding Java Bytecode executable instructions (reduced instruction set format). In addition, every one of the corresponding Java Bytecode executable instructions can be represented by at least one of the virtual machine instructions in the virtual machine instruction set.)

It would have been obvious to one ordinary skill in the art at the time the invention was made to combine Chan's method of producing secure executable code with Sokolov's method of producing reduced instruction set would further enhance Chan's method to produce the smaller instruction set from java bytecodes which would significantly reduce the amount of run time while executing the program.

In reference to claim 102:

Chen discloses:

*wherein the storage media is an optical disc on which the executable program, the security program and the emulation program are encoded.*

(Paragraph 44; A number of program modules may be stored on the hard disk, magnetic disk 216, optical disk 219, ROM 208 or RAM 210,

including an operating system 226, one or more application programs 228, other program modules 230, and program data 232.)

In reference to claim 103:

Chen discloses:

*wherein the optical disc is one of: a CD, a CD-ROM and a DVD.*

(Paragraph 42; The computer system 200 further includes a hard disk drive 212 for reading from and writing to a hard disk, a magnetic disk drive 214 for reading from or writing to a removable magnetic disk 216, and an optical disk drive 218 for reading from or writing to a removable optical disk 219 such as a CD ROM, DVD, or other optical media.)

In reference to claim 104:

Chen discloses:

*wherein the storage media is memory in, or associated with, servers, computers and/or other processing means and on which the security program and the emulation program are stored.*

(Paragraph 41; an exemplary computing system for embodiments of the invention includes a general purpose computing device in the form of a conventional computer system 200, including a processor unit 202, a system memory 204, and a system bus 206 that couples various system components including the system memory 204 to the processor unit 200. Paragraph 45; The computer system 200 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 246. The remote computer 246 may be a computer system, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer system 200.)

In reference to claim 105:

Art Unit: 2192

Chen discloses:
*wherein the executable program is a games program and/or a video program and/or an audio program, and/or other software.*
(Paragraph 44; Examples of other input devices may include a microphone, joystick, game pad, satellite dish, and scanner. For hand-held devices and tablet PC devices, electronic pen input devices may also be used. These and other input devices are often connected to the processing unit 202 through a serial port interface 240 that is coupled to the system bus 206. Nevertheless, these input devices also may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB).)

## Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to ZIAUL CHOWDHURY whose telephone number is (571)270-7750. The examiner can normally be reached on Monday Thru Friday, 7:30AM To 9:00PM, Alternet Friday, Eastern Time.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, TUAN Q. DAM can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Art Unit: 2192

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/ZIAUL CHOWDHURY/
Examiner, Art Unit 2192

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192